Robust Bichromatic Classification Using Two Lines

Erwin Glazenburg
Utrecht University, The Netherlands

Thijs van der Horst
Utrecht University, The Netherlands
TU Eindhoven, The Netherlands

Tom Peters
TU Eindhoven, The Netherlands

Bettina Speckmann
TU Eindhoven, The Netherlands

Frank Staals
Utrecht University, The Netherlands

Abstract

Given two sets R and B of n points in the plane, we present efficient algorithms to find a two-line linear classifier that best separates the "red" points in R from the "blue" points in B and is robust to outliers. More precisely, we find a region $\mathcal{W}_B$ bounded by two lines, so either a halfplane, strip, wedge, or double wedge, containing (most of) the blue points B, and few red points. Our running times vary between optimal $O(n \log n)$ up to around $O(n^3)$, depending on the type of region $\mathcal{W}_B$ and whether we wish to minimize only red outliers, only blue outliers, or both.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Geometric Algorithms, Separating Line, Classification, Bichromatic, Duality

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.33

Related Version *Full Version*: `https://arxiv.org/abs/2401.02897` [14]

1 Introduction

Let R and B be two sets of at most n points in the plane. Our goal is to best separate the "red" points R from the "blue" points B using at most two lines. That is, we wish to find a region $\mathcal{W}_B$ bounded by lines ℓ_1 and ℓ_2 containing (most of) the blue points B so that the number of points k_R from R in the interior $\text{int}(\mathcal{W}_B)$ of $\mathcal{W}_B$ and/or the number of points k_B from B in the interior of the region $\mathcal{W}_R = \mathbb{R}^2 \setminus \mathcal{W}_B$ is minimized. We refer to these subsets $\mathcal{E}_R = R \cap \text{int}(\mathcal{W}_B)$ and $\mathcal{E}_B = B \cap \text{int}(\mathcal{W}_R)$ as the red and blue outliers, respectively, and define $\mathcal{E} = \mathcal{E}_R \cup \mathcal{E}_B$ and $k = k_R + k_B$.

The region $\mathcal{W}_B$ is either: (i) a halfplane, (ii) a *strip* bounded by two parallel lines ℓ_1 and ℓ_2, (iii) a *wedge*, i.e., one of the four regions induced by a pair of intersecting lines ℓ_1 and ℓ_2, or (iv) a *double wedge*, i.e., two opposing regions induced by a pair of intersecting lines

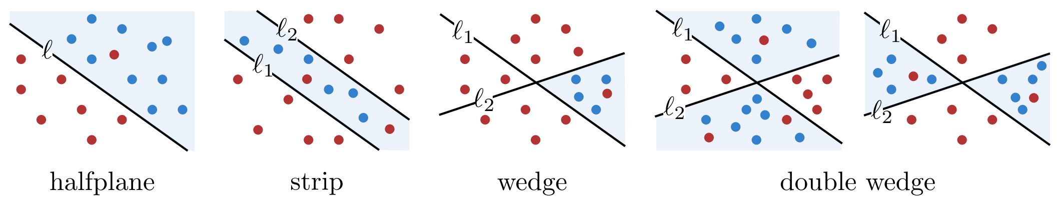

Figure 1 We consider separating R and B by at most two lines. This gives rise to four types of regions $\mathcal{W}_B$: halfplanes, strips, wedges, and two types of double wedges: hourglasses and bowties.

ℓ_1 and ℓ_2 (we further distinguish *hourglass* double wedges, that contain a vertical line, and the remaining *bowtie* double wedges), see Figure 1. We can reduce the case that $\mathcal{W}_B$ would consist of three regions to the single-wedge case by recoloring the points. For each of these cases for the shape of $\mathcal{W}_B$ we consider three problems: allowing only red outliers ($k_B = 0$) and minimizing k_R, allowing only blue outliers ($k_R = 0$) and minimizing k_B, or allowing both outliers and minimizing $k = k_R + k_B$. We present efficient algorithms for each of these problems, as shown in Table 1.

Motivation and related work. Classification is a key problem in computer science. The input is a labeled set of points and the goal is to obtain a procedure that, given an unlabeled point, assigns it a label that "fits it best", considering the labeled points. Classification has many direct applications, e.g. identifying SPAM in email messages, or tagging fraudulent transactions [20, 22], but is also the key subroutine in other problems such as clustering [1].

We restrict our attention to binary classification where our input is a set R of red points and a set B of blue points. We can compute whether R and B can be perfectly separated by a line (and compute such a line if it exists) in $O(n)$ time using linear programming. This extends to finding a separating hyperplane in case of points in $\mathbb{R}^d$, for some constant d [19].

Clearly, it is not always possible to find a hyperplane that perfectly separates the red and the blue points, see for example Figure 2, in which the blue points are actually all contained in a wedge. Hurtato et al. [16, 17] consider separating R and B in $\mathbb{R}^2$ using at most two lines ℓ_1 and ℓ_2. In this case, linear programming is unfortunately no longer applicable. Instead, they present $O(n \log n)$ time algorithms to compute a perfect separator (i.e., a strip, wedge, or double wedge containing all blue points but no red points), if it exists. These results were shown to be optimal [5], and can be extended to the case where B and R contain other geometric objects such as segments or circles, or to include constraints on the slopes [16]. Similarly, Hurtado et al. [18] considered similar strip and wedge separability problems for points in $\mathbb{R}^3$. Arkin et al. [4] show how to compute a 2-level binary space partition (a line ℓ and two rays starting on ℓ) separating R and B in $O(n^2)$ time, and a minimum height h-level tree, with $h \leq \log n$, in $n^{O(\log n)}$ time. Even today, computing perfect bichromatic separators with particular geometric properties remains an active research topic [2].

Alternatively, one can consider separation with a (hyper-)plane but allow for outliers. Chan [8] presented algorithms for linear programming in $\mathbb{R}^2$ and $\mathbb{R}^3$ that allow for up to k violations –and thus solve hyperplane separation with up to k outliers– that run in $O((n + k^2) \log n)$ and $O(n \log n + k^{11/4} n^{1/4} \operatorname{polylog} n)$ time, respectively. In higher (but constant) dimensions, only trivial solutions are known. For arbitrary (non-constant) dimensions the problem is NP-hard [3]. There is also a fair amount of work that aims to find a halfplane that minimizes some other error measure, e.g. the distance to the farthest misclassified point, or the sum of the distances to misclassified points [7, 15].

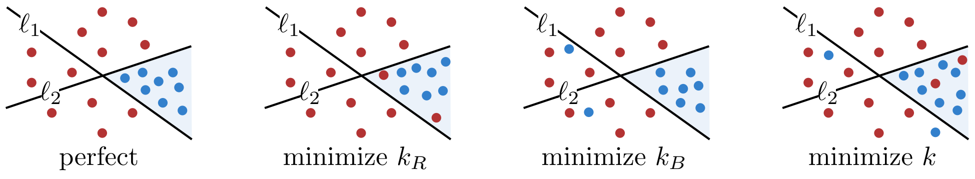

Figure 2 Perfectly separating R and B may require more than one line. When considering outliers, we may allow '(and minimize) only red outliers, only blue outliers, or both.

Table 1 An overview of our results. Results shown in the full version [14] are marked with "full". Expected running times are marked with a ‡.

region $\mathcal{W}_B$	minimize k_R		minimize k_B		minimize k	
halfplane	$O(n \log n)$	full	$O(n \log n)$	full	$O((n + k^2) \log n)$	[8]
strip	$O(n \log n)$	[21]	$O(n^2 \log n)$	full	$O(n^2 \log n)$	full
wedge	$O(n^2)$	[21]	$O(n^{5/2} \log n)$‡	§5.2		
	$O(n \log n)$	§5.1	$O(nk_B^2 \log^2 n \log k_B)$	§5.3	$O(nk^2 \log^3 n \log k)$	§5.3
double bowtie	$O(n^2)$	§6	$O(n^2 \log n)$	§6	$O(n^2 k \log^3 n \log k)$	§6

Separating points using more general non-hyperplane separators and outliers while incorporating guarantees on the number of outliers seems to be less well studied. Seara [21] showed how to compute a strip containing all blue points, while minimizing the number of red points in the strip in $O(n \log n)$ time. Similarly, he presented an $O(n^2)$ time algorithm for computing a wedge with the same properties. Armaselu and Daescu [6] show how to compute and maintain a smallest circle containing all red points and the minimum number of blue points. In this paper, we take some further steps toward the fundamental but challenging problem of computing a robust non-linear separator that provides performance guarantees.

Results. We present efficient algorithms for computing a region $\mathcal{W}_B = \mathcal{W}_B(\ell_1, \ell_2)$ defined by at most two lines ℓ_1 and ℓ_2 containing only the blue points, that are robust to outliers. Our results depend on the type of region $\mathcal{W}_B$ we are looking for, i.e., halfplane, strip, wedge, or double wedge, as well as on the type of outliers we allow: red outliers (counted by k_R), blue outliers (counted by k_B), or all outliers (counted by k). Refer to Table 1 for an overview.

Our main contributions are efficient algorithms for when $\mathcal{W}_B$ is really bounded by two lines. All these versions can be solved by a simple $O(n^4)$ time algorithm that explicitly considers all candidate regions. However, we show that we can do significantly better in every case.

In particular, in the versions where we minimize the number of red outliers k_R we achieve significant speedups. For example, we can compute an optimal wedge $\mathcal{W}_B$ containing B and minimizing k_R in optimal $\Theta(n \log n)$ time (which improves an earlier $O(n^2)$ time algorithm from Seara [21]). We use two types of duality transformations that allow us to map each point $p \in R \cup B$ into a *forbidden region* E_p in a low-dimensional parameter space, such that: *i)* every point s in this parameter space corresponds to a region $\mathcal{W}_B(s)$, and *ii)* this region $\mathcal{W}_B(s)$ misclassifies point p if and only if this point s lies in E_p. This allows us to solve the problem by computing a point that lies in the minimum number of forbidden regions.

Surprisingly, the versions of the problem in which we minimize the number of blue outliers k_B are much more challenging. For none of these versions we can match our running times for minimizing k_R, while needing more advanced tools. For example, for the single wedge version, we use dynamic lower envelopes to obtain a batched query problem that we solve using spanning-trees with low stabbing number [10]. See Section 5.2.

For the case where both red and blue outliers are allowed and we minimize k, we present output-sensitive algorithms whose running time depends on the optimal value of k. We essentially fix one of the lines ℓ_1, and use linear programming (LP) with violations [8, 13] to compute an optimal line ℓ_2 that together with ℓ_1 defines $\mathcal{W}_B$. We show that by using results on $\leq k$-levels, a recent data structure for dynamic LP with violations [13], and binary searching, we can achieve algorithms with running times around $O(n^2 k \operatorname{polylog} n)$.

Outline. We give some additional definitions and notation in Section 2, and in Section 3 we present a characterization of optimal solutions that lead to our simple $O(n^4)$ time algorithm for any type of wedges. In Sections 4, 5, and 6 we discuss the case when $\mathcal{W}_B$ is, respectively, a strip, wedge, or double wedge. In each of these sections we separately go over minimizing the number of red outliers k_R, the number of blue outliers k_B, and the total number of outliers k. We wrap up with some concluding remarks and future work in Section 7. Omitted proofs can be found in the full version [14].

2 Preliminaries

In this section we discuss some notation and concepts used throughout the paper. For ease of exposition we assume $B \cup R$ contains at least three points and is in general position, i.e., that all coordinate values are unique, and that no three points are colinear.

Notation. Let ℓ^- and ℓ^+ be the two halfplanes bounded by line ℓ, with ℓ^- below ℓ (or left of ℓ if ℓ is vertical). Any pair of lines ℓ_1 and ℓ_2, with the slope of ℓ_1 smaller than that of ℓ_2, subdivides the plane into at most four interior-disjoint regions: $\text{North}(\ell_1, \ell_2) = \ell_1^+ \cap \ell_2^+$, $\text{East}(\ell_1, \ell_2) = \ell_1^+ \cap \ell_2^-$, $\text{South}(\ell_1, \ell_2) = \ell_1^- \cap \ell_2^-$ and $\text{West}(\ell_1, \ell_2) = \ell_1^- \cap \ell_2^+$. When ℓ_1 and ℓ_2 are clear from the context we may simply write North to mean $\text{North}(\ell_1, \ell_2)$, etc. We assign each of these regions to either B or R, so that $\mathcal{W}_B = \mathcal{W}_B(\ell_1, \ell_2)$ and $\mathcal{W}_R = \mathcal{W}_R(\ell_1, \ell_2)$ are the union of some elements of $\{\text{North}, \text{East}, \text{South}, \text{West}\}$. In case ℓ_1 and ℓ_2 are parallel, we assume that ℓ_1 lies below ℓ_2, and thus $\mathcal{W}_B = \text{East}$.

Duality. We make frequent use of the standard point-line duality [11], where we map objects in *primal* space to objects in a *dual* space. In particular, a primal point $p = (a, b)$ is mapped to the dual line $p^* : y = ax - b$ and a primal line $\ell : y = ax + b$ is mapped to the dual point $\ell^* = (a, -b)$. If in the primal a point p lies above a line ℓ, then in the dual the line p^* lies below the point ℓ^*.

For a set of points P with duals $P^* = \{p^* \mid p \in P\}$, we are often interested in the *arrangement* $\mathcal{A}(P^*)$, i.e., the vertices, edges, and faces formed by the lines in P^*. Two unbounded faces of $\mathcal{A}(P^*)$ are *antipodal* if their unbounded edges have the same two supporting lines. Since every line contributes to four unbounded faces, there are $O(n)$ pairs of antipodal faces. We denote the upper envelope of P^*, i.e., the polygonal chain following the highest line in $\mathcal{A}(P^*)$, by $\mathcal{U}(P^*)$, and the lower envelope by $\mathcal{L}(P^*)$.

3 Properties of an optimal separator

Next, we prove some structural properties about the lines bounding the region $\mathcal{W}_B$ containing (most of) the blue points in B.

▶ **Lemma 3.1.** *For the strip classification problem there exists an optimum where one line goes through two points and the other through at least one point.*

Proof. Clearly, we can shrink an optimal strip $\mathcal{W}_B(\ell_1, \ell_2)$ so that both ℓ_1 and ℓ_2 contain a (blue) point, say b_1 and b_2, respectively. Now rotate ℓ_1 around b_1 and ℓ_2 around b_2 in counter-clockwise direction until either ℓ_1 or ℓ_2 contains a second point. ◀

▶ **Lemma 3.2.** *For any wedge classification problem there exists an optimum where both lines go through a blue and a red point.*

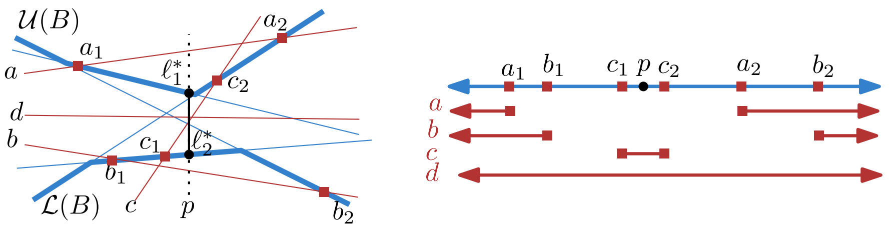

Figure 3 The four types of red lines and their forbidden region.

Proof sketch. Similar to the proof for strips, we show that any (double) wedge can be adjusted until both its lines go through a blue and a red point, without misclassifying any more points. Since this also holds for a given optimum, the lemma follows. ◀

A simple general algorithm. Lemma 3.2 tells us we only have to consider lines through red and blue points. Hence, there is a simple brute-force $O(n^4)$ time algorithm that considers all pairs of such lines, which works for both wedges and double wedges and any type of outliers. Refer to the full version for details.

4 Separation with a strip

In this section we consider the case where lines ℓ_1 and ℓ_2 are parallel, with ℓ_2 above ℓ_1, and thus $\mathcal{W}_B(\ell_1, \ell_2)$ forms a strip. We want B to be inside the strip, and R outside. We work in the dual, where we want to find two points ℓ_1^* and ℓ_2^* with the same x-coordinate such that vertical segment $\overline{\ell_1^*\ell_2^*}$ intersects the lines in B^* but not the lines in R^*. We briefly summarize our approach and our results.

Strip separation with red outliers. In this version, we wish to find a vertical line segment $\overline{\ell_1^*\ell_2^*}$ that intersects all lines in B^* and minimizes the number of lines from R^* it intersects. So we can assume that ℓ_1^* lies on the upper envelope $\mathcal{U}(B^*)$ of B^* and ℓ_2^* lies on the lower envelope $\mathcal{L}(B^*)$, since shortening $\overline{\ell_1^*\ell_2^*}$ can only decrease the number of red lines intersected. There is only one degree of freedom for choosing our segment, its x-coordinate, so our *parameter space* is $\mathbb{R}$. We parameterize $\mathcal{U}(B^*)$ and $\mathcal{L}(B^*)$ over $\mathbb{R}$, so that each point $p \in \mathbb{R}$ in the parameter space corresponds to the vertical segment $\overline{\ell_1^*\ell_2^*}$ on the line $x = p$. See Figure 3. We map every red line r to a forbidden region (an interval) in this parameter space, in which $\overline{\ell_1^*\ell_2^*}$ would intersect r. Our goal is then to compute a point p that is contained in the minimum number of such forbidden intervals. We can do so in $O(n \log n)$ time by sorting and scanning. This matches an existing result by Seara [21].

Strip separation with blue outliers. In this version, the vertical segment $\overline{\ell_1^*\ell_2^*}$ may intersect no red lines and as many blue lines as possible. That means that there is a $\overline{\ell_1^*\ell_2^*}$ that is a maximal vertical segment in a face of $\mathcal{A}(R^*)$. We sweep a vertical line ℓ over $\mathcal{A}(R^*)$ while, for each face F (defining a candidate segment $F \cap \ell$ for $\overline{\ell_1^*\ell_2^*}$) we maintain the number of blue lines that intersect the candidate segment. This leads to an $O(n^2 \log n)$ time algorithm for computing an optimal segment, and thus an optimal strip.

Strip separation with both outliers. In this version, the vertical segment $\overline{\ell_1^*\ell_2^*}$ may misclassify both red and blue lines, but as few as possible. There is much less structure for where an optimal segment can be than before, since an optimal segment can now intersect any

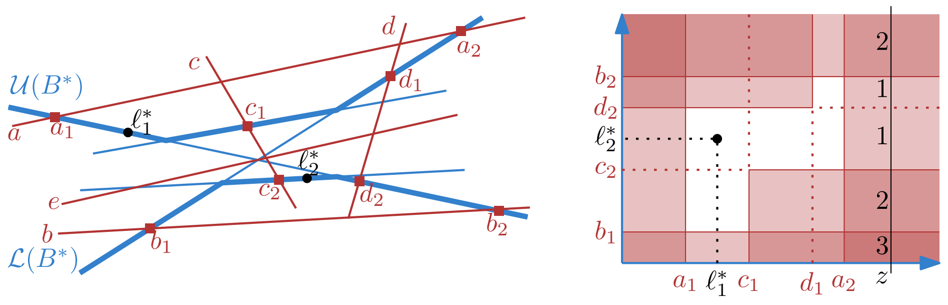

Figure 4 The arrangement of $B^* \cup R^*$ with its parameter space and forbidden regions.

number of blue or red lines. We sweep over the full arrangement $\mathcal{A}(R^* \cup B^*)$ with a vertical line. We maintain a datastructure that, given a point ℓ_2^* on the sweepline, can find a second point ℓ_1^* such that the number of outliers $|\mathcal{E}(\ell_1, \ell_2)|$ is minimized. Each time the sweepline crosses a vertex of $\mathcal{A}(R^* \cup B^*)$ we update the datastructure and perform one query, both in $O(\log n)$ time, resulting in an $O(n^2 \log n)$ algorithm.

▶ **Theorem 4.1.** *Given two sets of n points $B, R \subset \mathbb{R}^2$, we can construct a strip $\mathcal{W}_B$ minimizing (i) the number of red outliers k_R in $O(n \log n)$ time, (ii) the number of blue outliers k_B in $O(n^2 \log n)$ time, or (iii) the number of outliers k in $O(n^2 \log n)$ time.*

5 Separation with a wedge

We consider the case where the region $\mathcal{W}_B$ is a single wedge and $\mathcal{W}_R$ is the other three wedges. In Sections 5.1, 5.2, and 5.3 we show how to minimize k_R, k_B, and k, respectively.

5.1 Wedge separation with red outliers

We distinguish between $\mathcal{W}_B$ being an East or West wedge, and a North or South wedge. In either case we can compute optimal lines ℓ_1 and ℓ_2 defining $\mathcal{W}_B$ in $O(n \log n)$ time.

Finding an East or West wedge. We wish to find two lines ℓ_1 and ℓ_2 such that every blue point and as few red points as possible lie above ℓ_1 and below ℓ_2. In the dual this corresponds to points ℓ_1^* and ℓ_2^* such that all blue lines and as few red lines as possible lie below ℓ_1^* and above ℓ_2^*, as in Figure 4.

Clearly ℓ_1^* must lie above $\mathcal{U}(B^*)$, and ℓ_2^* below $\mathcal{L}(B^*)$, and again we can assume they lie on $\mathcal{U}(B^*)$ and $\mathcal{L}(B^*)$, respectively. We now have two degrees of freedom, one for choosing ℓ_1^* and one for choosing ℓ_2^*. Again we parameterize $\mathcal{U}(B^*)$ and $\mathcal{L}(B^*)$, but this time over $\mathbb{R}^2$, such that a point (p, q) in this parameter space corresponds to two dual points ℓ_1^* and ℓ_2^*, with ℓ_1^* on $\mathcal{U}(B^*)$ at $x = p$ and $\ell_2^*(y)$ on $\mathcal{L}(B^*)$ at $x = q$, as illustrated in Figure 4. We wish to find a value in our parameter space whose corresponding segment minimizes the number of red misclassifications. Recall the forbidden regions of a red line r are those regions in the parameter space in which corresponding segments misclassify r. We distinguish between five types of red lines, as in Figure 4:

- Line a intersects $\mathcal{U}(B^*)$ in points a_1 and a_2, with a_1 left of a_2. Only segments with ℓ_1^* left of a_1 or right of a_2 misclassify a. This produces two forbidden regions: $(-\infty, a_1) \times (-\infty, \infty)$ and $(a_2, \infty) \times (-\infty, \infty)$.
- Line b intersects $\mathcal{L}(B^*)$ in points b_1 and b_2, with b_1 left of b_2. Symmetric to line a this produces forbidden regions $(-\infty, \infty) \times (-\infty, b_1)$ and $(-\infty, \infty) \times (b_2, \infty)$.

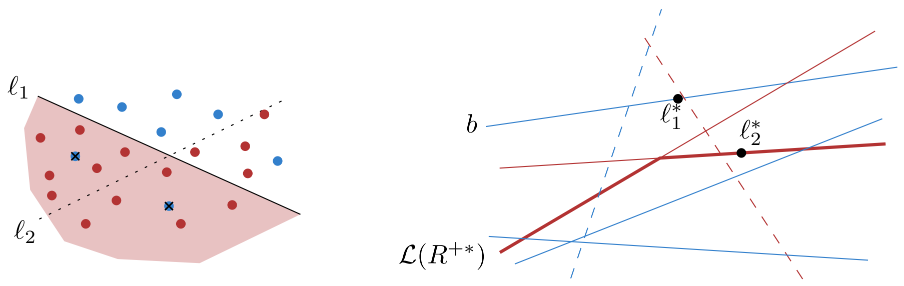

Figure 5 Left: in the primal we need to consider only the points above ℓ_1. Right: in the dual we need to consider only the lines below ℓ_1^*. In particular, ℓ_1^* should lie below (on) $\mathcal{L}(R^{+*})$.

- Line c intersects $\mathcal{U}(B^*)$ in c_1 and $\mathcal{L}(B^*)$ in c_2, with c_1 left of c_2. Only segments with endpoints after c_1 and before c_2 misclassify c. This produces the region $(c_1, \infty) \times (-\infty, c_2)$. (Segments with endpoints before c_1 and after c_2 do intersect c, but do not misclassify it.)
- Line d intersects $\mathcal{U}(B^*)$ in d_1 and $\mathcal{L}(B^*)$ in d_2, with d_1 right of d_2. Symmetric to line c it produces the forbidden region $(-\infty, d_1) \times (d_2, \infty)$.
- Line e intersects neither $\mathcal{U}(B^*)$ nor $\mathcal{L}(B^*)$. All segments misclassify e. In the primal this corresponds to red points inside the blue convex hull. This produces one forbidden region; the entire plane $\mathbb{R}^2$.

The forbidden regions generated by the red lines $r^* \in R^*$ divide the parameter space in axis-aligned orthogonal regions. Our goal is again to find a point with minimum *ply*, i.e. a point that is contained in the minimum number of these forbidden regions.

▶ **Lemma 5.1.** *Given a set $\mathcal{R}$ of n constant complexity, axis-aligned, orthogonal regions, we can compute the point with minimum ply in $O(n \log n)$ time.*

Proof sketch. We sweep through the plane with a vertical line z while maintaining a minimum ply point on z. See Figure 4 (right) for an illustration. We maintain the regions intersected by the sweep line in a slightly augmented segment tree [11]. In particular, each node v in the tree stores the size $s(v)$ of its canonical subset, the minimum ply $\text{ply}(v)$ within the subtree of v, and a point attaining this minimum ply. Since there are $O(n)$ forbidden regions (rectangles), each of which is added and removed once in $O(\log n)$ time, this leads to a running time of $O(n \log n)$. ◀

We construct $\mathcal{U}(B^*)$ and $\mathcal{L}(B^*)$ in $O(n \log n)$ time. For every red line r, we calculate its intersections with $\mathcal{U}(B^*)$ and $\mathcal{L}(B^*)$ in $O(\log n)$ time, determine its type $(a - e)$, and construct its forbidden regions. By Lemma 5.1 we can find a point with minimum ply in these forbidden regions in $O(n \log n)$ time. We thus obtain an $O(n \log n)$ time algorithm for finding an optimal East or West wedge. We can find an optimal North or South wedge in a similar manner, and thus obtain:

▶ **Theorem 5.2.** *Given two sets of n points $B, R \subset \mathbb{R}^2$, we can construct a wedge $\mathcal{W}_B$ containing all points of B and the fewest points of R in $O(n \log n)$ time.*

5.2 Wedge separation with blue outliers

We now consider the case where all red points must be classified correctly, and we minimize the number of blue outliers k_B. We show how to find an optimal North wedge; finding optimal South, East, or West wedges can be done analogously.

Fix line ℓ_1 and consider the problem of finding an optimal corresponding line ℓ_2. All points below ℓ_1 lie outside the North wedge, regardless of our choice of ℓ_2, and thus we have to consider only the points $B^+ \subseteq B$ and $R^+ \subseteq R$ above ℓ_1. See Figure 5. We do not allow red points in the North wedge, so ℓ_2 must lie above all red points R^+ and below as many blue points B^+ as possible. This is exactly the halfplane separation problem with blue outliers we solve in the full version in $O(n \log n)$ time. We can iterate through all $O(n^2)$ options for ℓ_1 (by walking through $\mathcal{A}(B^* \cup R^*)$) and compute the corresponding line ℓ_2 in $O(n \log n)$ time, which would lead to an $O(n^3 \log n)$ time algorithm. Below we describe an algorithm that avoids recomputing ℓ_2 from scratch every time, giving an $O(n^{5/2} \log n)$ time algorithm.

Let L be a set of lines, and let the *level* $l_L(p)$ of a point p (with respect to L) be the number of lines of L that lie below p. We define the level $l_L(s) = \max_{p \in s} l_L(p)$ of a segment s (with respect to L) as the maximum level of any point p on s.

Consider the dual, where we are looking for two points ℓ_1^* and ℓ_2^* such that no red line and as many blue lines as possible lie below both ℓ_1^* and ℓ_2^*. By Lemma 3.2 we can assume both ℓ_1^* and ℓ_2^* lie on a red-blue intersection. For a fixed point ℓ_1^* we are interested only in the set of lines B^{+*} and R^{+*} below ℓ_1^*, and since ℓ_2^* must lie below all of R^{+*} we can assume it lies on its lower envelope $\mathcal{L}(R^{+*})$. See Figure 5. The wedge North(ℓ_1, ℓ_2) correctly classifies exactly $l_{B^{+*}}(\ell_2^*)$ points. We are thus looking for the pair of points ℓ_1^* and ℓ_2^* that maximize the level $l_{B^{+*}}(\ell_2^*)$.

We now show that we can compute ℓ_2^* efficiently for every candidate point ℓ_1^*, provided that there is an oracle that can answer (a batch of) the following queries: given a point ℓ_1^* and a line segment s lying on a red line, compute the level $l_{B^{+*}}(s)$. We then show that we can implement an oracle that answers all $O(n^2)$ queries in $O(n^{5/2} \log n)$ time. This yields an $O(n^{5/2} \log n)$ time algorithm to compute an optimal north wedge.

Using an oracle to maintain ℓ_2^*. Consider any blue line $b \in B^*$ and assume w.l.o.g. that it is horizontal. We will shift ℓ_1^* from left to right along b, maintaining the set of red lines R^{+*} below ℓ_1^*. During this shift ℓ_1^* crosses each of the other lines at most once. We wish to maintain $\mathcal{L}(R^{+*})$ and a point with maximum level w.r.t. B^{+*} over all edges of $\mathcal{L}(R^{+*})$. Such a point corresponds to an optimal second point ℓ_2^* for the current point ℓ_1^*. By repeating this shift for every blue line $b \in B^*$ we consider all $O(n^2)$ candidate points for ℓ_1^* and their corresponding optimal point ℓ_2^*. This thus allows us to report an optimal solution.

We first show that we may keep an explicit representation of $\mathcal{L}(R^{+*})$ while shifting ℓ_1^* along b. We store the edges of $\mathcal{L}(R^{+*})$ in the leaves of a binary tree (ordered on increasing x-coordinate), which we refer to as the *explicit tree* of $\mathcal{L}(R^{+*})$. We then augment this explicit tree to additionally maintain the maximum level over all its edges. We show that we can maintain this tree in near-linear time in total; see the full version for details.

▶ **Lemma 5.3.** *While shifting ℓ_1^* along b we can maintain an explicit tree of $\mathcal{L}(R^{+*})$ in $O(n \log n)$ total time.*

With the explicit tree at hand, we require only $O(n)$ queries to the oracle during the entire shifting process to maintain an optimal point ℓ_2^*. Refer to the full version for details.

▶ **Lemma 5.4.** *We can maintain an edge of $\mathcal{L}(R^{+*})$ with maximum level w.r.t. B^{+*} while shifting ℓ_1^* along b using $O(n)$ queries to the oracle and $O(n \log n)$ additional time.*

Collecting queries to the oracle. What remains is to describe how to implement the oracle that answers the queries. Observe that the set of queries to the oracle is fixed. That is, the answer to an oracle query is independent of the answers to earlier queries, and the answers of queries do not influence which future queries will be performed. Therefore, we can perform a "dry"-run of the algorithm where we collect all queries, and then answer them in bulk. As we will see, this allows us to answer these queries efficiently.

Since each blue line generates $O(n)$ queries (Lemma 5.4), we have a total of $O(n^2)$ queries. Once we have the answers to all these queries, we once again run the algorithm for each blue line b. In this "real"-run of the algorithm we can answer queries in $O(1)$ time, and thus compute an optimal pair of points ℓ_1^* and ℓ_2^* with ℓ_1^* on b in $O(n \log n)$ time (Lemma 5.4). This then leads the following result.

▶ **Lemma 5.5.** *Given two sets of n points $B, R \subset \mathbb{R}^2$, we can construct a wedge containing as many points of B as possible and no points of R in $O(T(n) + n^2 \log n)$ time, where $T(n)$ is the total time required to answer $O(n^2)$ oracle queries.*

Implementing the oracle. We will now show how to implement the oracle that can answer (a batch of) the following queries efficiently. Given a query (ℓ_1^*, s) consisting of a point ℓ_1^* and a red segment s, we wish to find the maximum level of any point on s w.r.t. the set B^{+*} of blue lines below ℓ_1^*. Maintaining the set B^{+*} and answering queries fully dynamically is difficult, so we will instead answer them in bulk.

We consider a red line r and assume w.l.o.g. that it is horizontal. Let Q be the set of queries whose line segment s lies on r, let $q_r = |Q|$, and let P be the set of query points ℓ_1^* corresponding to the queries in Q. See Figure 6.

We pick an arbitrary query $(\ell_1^*, s) \in Q$. Let $b \in B^{+*}$ be a blue line with negative slope; the other case is analogous. Consider the intersection point i between b and r. For points $p \in s$ left of i, b lies above p and thus b does not add to the level of p. For points $p \in s$ right of i, b lies below i and thus b does add to the level of p. Consider all intersections between r and lines in B^{+*}. We build a balanced binary tree on these intersections, ordered by x-coordinate, augmented such that each node also stores the point with the highest level in its subtree. Recall that s is a line segment on r, and thus represents an x-interval on r. We can easily answer the query (s, ℓ_1^*) by finding the $O(\log n)$ nodes representing that interval and returning the maximum level of any point inside their subtrees.

To answer the other queries we can shift the point ℓ_1^* through the arrangement $\mathcal{A}(B^*)$. When we cross into a different face, one blue line is inserted into or deleted from B^{+*}. We can update the binary tree in $O(\log n)$ time, meaning that when we reach a point $p \in P$ we can answer the corresponding query, again in $O(\log n)$ time. So, if we can walk through $\mathcal{A}(B^*)$ crossing s lines while visiting all points P, we can answer all queries Q in $O((s + |P|) \log n)$ time.

Consider a spanning tree on P. The stabbing number of a spanning tree is the maximum number of edges of the tree that can be intersected by a single line. With high probability (whp; in particular with probability $1 - 1/q_r^c$ for some arbitrarily large constant c) we can build a spanning tree T on P with stabbing number $O(\sqrt{q_r})$ in $O(q_r \log q_r)$ time [9]. Thus, each blue line intersects T at most $O(\sqrt{q_r})$ times, and therefore there are $O(n\sqrt{q_r})$ intersections between T and B (whp).

If we follow the spanning tree while walking through $\mathcal{A}(B^*)$ we will thus cross $O(n\sqrt{q_r})$ blue lines in total while visiting all points P, meaning we can answer all queries Q on r in $O(n\sqrt{q_r} \log n)$ time (note that the $O(q_r \log q_r)$ time to build the spanning tree is dominated by $O(n\sqrt{q_r} \log n)$ for any $q_r = O(n^2)$). Doing this for all lines $r \in R$ takes $O(\sum_r (n\sqrt{q_r} \log n))$ time. Recall we have $O(n^2)$ queries in total, so we have $\sum_r q_r = O(n^2)$. Since $\sqrt{\cdot}$ is a concave function, we have $\sqrt{a} + \sqrt{b} \leq \sqrt{2(a+b)}$ for any non-negative values a and

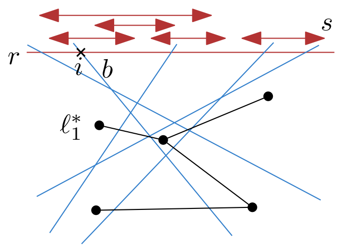

Figure 6 A set of queries on a red line r, with a spanning tree on P. Line b contributes to the level of all points right of i.

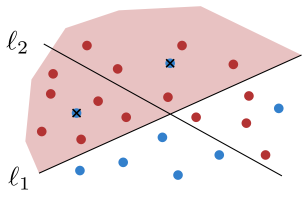

Figure 7 All points above ℓ_1 lie outside the South wedge. After fixing ℓ_1, we are left with a halfplane separation problem.

b. More generally, $\sum_i \sqrt{x_i} \le \sqrt{n\,(\sum_i x_i)}$ for any n non-negative values x_i. In particular, $\sum_r \sqrt{q_r} \le \sqrt{n\,(\sum_r q_r)} = O(n^{3/2})$. Therefore, we have an $O(\sum_r (n\sqrt{q_r}\log n)) = O(n^{5/2}\log n)$ time algorithm to answer all queries over all red lines.

▶ **Lemma 5.6.** *We can answer $O(n^2)$ queries in expected $O(n^{5/2}\log n)$ time.*

Together with Lemma 5.5 this yields an expected $O(n^{5/2}\log n + n^2\log n) = O(n^{5/2}\log n)$ time algorithm to find an optimal North wedge. We can symmetrically find an optimal South wedge by assuming the wedge lies below both ℓ_1 and ℓ_2. Similarly by assuming the wedge lies below ℓ_1 and above ℓ_2 we can find an optimal West or East wedge.

▶ **Theorem 5.7.** *Given two sets of n points $B, R \subset \mathbb{R}^2$, we can construct a wedge containing as many points of B as possible and no points of R in expected $O(n^{5/2}\log n)$ time.*

5.3 Wedge separation with both outliers

We now consider the case where we allow and minimize both red and blue outliers. We show how to find an optimal South wedge; finding an optimal West, North, or East wedge can be done symmetrically. We first study the decision version of this problem: given an integer k', does there exist a South wedge $\mathcal{W}_B$ with at most k' outliers? We present an $O(nk'^2\log^3 n)$ time algorithm to solve this decision problem. Using exponential search to guess the optimal value k (i.e. guessing $k' = 1, 2, 4, 8\dots$ and binary searching in the remaining interval) then leads to an $O(nk^2\log^3 n\log k)$ time algorithm to compute a wedge $\mathcal{W}_B$ that minimizes k.

In Lemma 5.8 we construct a small candidate set of lines that contains a line ℓ_1 that is used by an optimal wedge. Then our algorithm considers each line in this set and constructs an optimal wedge using it.

▶ **Lemma 5.8.** *In $O(nk'\log n)$ time, we can construct a set of $O(nk')$ lines that contains a line ℓ_1 used by an optimal wedge.*

Proof. Consider any line ℓ and suppose it is used in a South wedge as the line ℓ_1. Let B^+ and B^- be the set of blue points above, respectively below, ℓ_1. Since we are looking for a South wedge, all $k_1 = |B^+|$ blue points above ℓ_1 are misclassified, regardless of line ℓ_2 (see Figure 7). Therefore, any line with $k_1 > k'$ blue points above it is not a suitable candidate for ℓ_1. In the dual plane, this means ℓ_1^* must lie in the $(\le k')$*-level* $L_{\le k'}(B^*)$ *of* B^*, the set of points with at most k' lines of B^* below them. With a slight abuse of notation, we use $L_{\le k'}(B^*)$ to refer to the sub-arrangement of $\mathcal{A}(B^*)$ that lies in the $(\le k')$-level. The complexity of $L_{\le k'}(B^*)$ is $O(nk')$, and we can construct $L_{\le k'}(B^*)$ in $O(nk' + n\log n)$ time [12].

Consider any line $r \in R^*$, and observe that it intersects $L_{\leq k'}(B^*)$ at most $O(k')$ times. This follows from the fact that we can decompose $L_{\leq k'}(B^*)$ into $O(k')$ concave chains [8], and that r intersects each such chain at most twice. We can thus explicitly compute all the $O(nk')$ red-blue intersections in $L_{\leq k'}(B^*)$ in $O(nk' \log n)$ time, and by Lemma 3.2 these red-blue intersections contain the dual of a line used in an optimal wedge. ◀

Fix ℓ_1 to be any candidate line from Lemma 5.8. We wish to find another line ℓ_2 such that the wedge South(ℓ_1, ℓ_2) misclassifies at most k' blue points. Since all points above ℓ_1 are outside the wedge regardless of the line ℓ_2, we need to consider only the points B^- and R^- below ℓ_1. Recall that the choice of ℓ_1 already misclassifies k_1 blue points. Thus, we wish to find a line ℓ_2 that misclassifies at most $k_2 = k' - k_1$ points from B^- and R^-. That is, a line ℓ_2 such that the number of points from R^- below it plus the number of points from B^- above it is at most k_2. This is exactly the halfplane separation problem with both outliers, which we can solve using Chan's algorithm in $O((n + (k_2)^2) \log n)$ time [8]. Doing this for all $O(nk')$ candidate lines results in an $O(nk'(n + k^2) \log n) = O((n^2k' + nk'^3) \log n)$ time algorithm. Below we improve on this, by avoiding to recompute ℓ_2 from scratch every time.

Solving the halfplane separation problem dynamically. Consider again the set of candidate lines of Lemma 5.8, and in particular their dual points. By walking through the arrangement $L_{\leq k'}(B^*)$, we can visit all $O(nk')$ candidate points ℓ_1^* in $O(nk')$ steps, such that at each step we cross only one (red or blue) line. This means that only a single point is inserted in or deleted from the sets B^- and R^- per step. Rather than computing ℓ_2 from scratch after every step now, we maintain it dynamically.

We build the data structure of [13] that, given a value k', maintains a line ℓ_2 that misclassifies as few points from R^- and B^- as possible under insertions and deletions of red and blue points; if no line misclassifying at most k' points exists, the data structure reports this. The updates for the data structure have to be given in a 'semi-online' manner, which means that whenever a point is inserted we have to know when it is going to be deleted. This is not a problem in our case, since we can precompute all insertions and deletions on R^- and B^- by completing the walk through $L_{\leq k'}(B^*)$ before actually updating the data structure.

The data structure reports an optimal line ℓ_2 that misclassifies $k_2 \leq k'$ points of R^- and B^-, so the wedge South(ℓ_1, ℓ_2) then misclassifies $k_1 + k_2$ points. If $k_1 + k_2 \leq k'$ then we have found a wedge misclassifying at most k' points, so we are done with the decision problem, otherwise we move on to the next candidate for ℓ_1. If the data structure reports that there exists no line ℓ_2 misclassifying at most k' points, then we also move on to the next candidate.

The data structure has update time $O(k' \log^3 n)$ per insertion and deletion, and there are $O(nk')$ updates. Therefore, given a value k', we can find a South wedge with at most k' outliers, if it exists, in $O(nk'^2 \log^3 n)$ time. Using exponential search for the optimal value k then gives an optimal South wedge in $O(nk^2 \log^3 n \log k)$ time. As in the previous section, we can similarly find North, West, and East wedges.

▶ **Theorem 5.9.** *Given two sets of n points $B, R \subset \mathbb{R}^2$, we can construct a wedge $\mathcal{W}_B$ minimizing the total number of outliers k in $O(nk^2 \log^3 n \log k)$ time.*

With similar techniques to the above, we can improve (for some values of k_B) our algorithm for allowing blue outliers only to have an output-sensitive running time, see the full version for details.

▶ **Theorem 5.10.** *Given two sets of n points $B, R \subset \mathbb{R}^2$, we can construct a wedge $\mathcal{W}_B$ minimizing the number of blue outliers k_B in $O(nk_B^2 \log^2 n \log k_B)$ time.*

6 Separation with a double wedge

The final setting we study is that of finding a double wedge. We summarize our approach and results for all three cases.

Double wedge separation with red outliers. We consider finding a *bowtie* wedge $\mathcal{W}_B$ while minimizing red outliers, i.e. all of B and as little of R as possible lies in the West and East wedge. In the dual this corresponds to a line segment intersecting all of B^*, and as little of R^* as possible.

Observe that a segment intersecting all lines of B^* must have endpoints in antipodal outer faces of $\mathcal{A}(B^*)$, i.e. two opposite outer faces sharing the same two infinite bounding lines. For all $O(n)$ pairs of antipodal faces, we could apply a very similar algorithm to the wedge algorithm in Section 5.1, resulting in $O(n \cdot n \log n) = O(n^2 \log n)$ time.

Alternatively, we construct the entire arrangement $\mathcal{A}(B^* \cup R^*)$ of all lines explicitly in $O(n^2)$ time (see e.g. [11]). Consider a pair of faces P and Q that are antipodal in $\mathcal{A}(B^*)$, and assume w.l.o.g. they are separated by the x-axis, with P above Q. There are two types of red lines: *splitting* lines that intersect both P and Q once, and *stabbing* lines that intersect at most one of P and Q, see Figure 8. A red line is a splitting line for exactly one pair of antipodal faces, while it can be a stabbing line for multiple pairs. Recall that we wish to find a segment from P to Q intersecting as few red lines as possible. The s splitting lines divide the boundary of P and Q into $s+1$ chains $P_0..P_s$ ($Q_0..Q_s$). Within one such chain P_i on P we only need to consider the point p_i with the most stabbing lines above it: a segment from p_i to Q will not intersect those lines, since Q is below P_i. Similarly, we only need to consider point q_j on chain Q_j with the most stabbing lines below it. Using dynamic programming we can then find the pair of chains P_i, Q_j such that $\overline{p_i q_j}$ intersects the fewest red lines in $O(n + s^2)$ time. Doing so for all pairs of antipodal faces yields a total running time of $O(n^2)$.

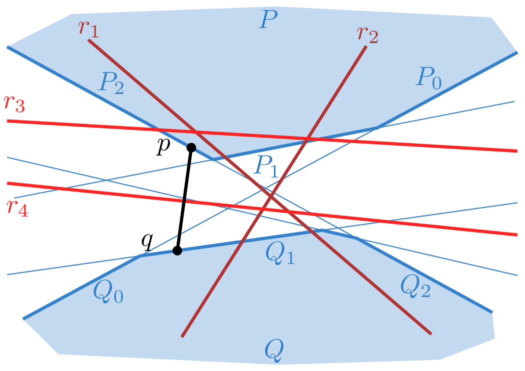

Figure 8 Two antipodal faces P and Q, with two splitting lines r_1, r_2 and two stabbing lines r_3, r_4, and an optimal segment $\overline{pq}$ from P to Q.

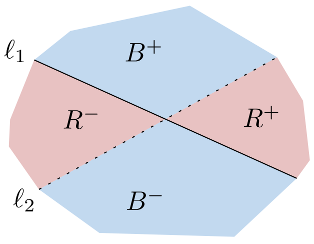

Figure 9 If we want B to lie in the North and South wedges, then B^+ and R^- should be above ℓ_2, and B^- and R^+ should be below ℓ_2.

▶ **Theorem 6.1.** *Given two sets of n points $B, R \subset \mathbb{R}^2$, we can construct the bowtie double wedge $\mathcal{W}_B$ minimizing the number of red outliers k_R in $O(n^2)$ time.*

Double wedge separation with blue outliers. We wish to find a bowtie wedge $\mathcal{W}_B$ while minimizing blue outliers, i.e. none of R and as much of B as possible should lie in the West and East wedge. In the dual this corresponds to a line segment intersecting none of R^*, and as much of B^* as possible. This means the segment must lie in a single face of $\mathcal{A}(R^*)$. For each face F of $\mathcal{A}(R^*)$ we thus wish to find a segment intersecting as many blue lines as possible. Let B^*_F be the set of blue lines intersecting F. Then we can find such a segment in $O(|B^*_F| \log |B^*_F|)$ time with a parameter space approach similar to Section 5.1. We do this for each face F in $\mathcal{A}(R^*)$, yielding an $O(n^2 \log n)$ algorithm.

Double wedge separation with both outliers. We show how to find an hourglass wedge $\mathcal{W}_B$ while minimizing both types of outliers; by recolouring we can also find an optimal bowtie wedge. We use a similar approach as in Section 5.2, by considering a set of $O(n^2)$ candidate lines ℓ_1 and computing an optimal line ℓ_2 for each. For a fixed line ℓ_1, let B^+ and R^+ be the points above ℓ_1, and B^- and R^- be the points below ℓ_1, and let $P = B^+ \cup R^-$ and $Q = B^- \cup R^+$. See Figure 9. Then we wish to find a line ℓ_2 separating P and Q. Using the same dynamic datastructure for halfplane separation as used in Section 5.3, we can compute an optimal ℓ_2 for each ℓ_1 in $O(n^2 k \log^3 n \log k)$ time.

7 Concluding Remarks

We presented efficient algorithms for robust bichromatic classification of $R \cup B$ with at most two lines. Our results depend on the shape of the region containing (most of the) blue points B, and whether we wish to minimize the number of red outliers, blue outliers, or both. See Table 1. Several of our algorithms reduce to the problem of computing a point with minimum ply with respect to a set of regions. We can extend these algorithms to support weighted regions, and thus we may support classifying weighted points (minimizing the weight of the misclassified points). It is interesting to see if we can support other error measures as well.

There are also still many interesting open questions. Most prominently whether we can obtain faster algorithms for minimizing the number of blue outliers k_B or the total number of outliers k. Alternatively, it would be interesting to establish lower bounds for the various problems. In particular, are our algorithms for computing a halfplane minimizing k_R optimal, and in case of wedges (where the problem is asymmetric) is minimizing the number of blue outliers k_B really more difficult then minimizing k_R? For the strip case, the running time of our algorithm for minimizing k matches the worst case running time for halfplanes ($O((n + k^2) \log n)$, which is $O(n^2 \log n)$ when $k = O(n)$), but it would be interesting to see if we can also obtain algorithms sensitive to the number of outliers k.

References

1 Charu C. Aggarwal, editor. *Data Classification: Algorithms and Applications.* CRC Press, 2014. doi:10.1201/B17320.

2 Carlos Alegría, David Orden, Carlos Seara, and Jorge Urrutia. Separating bichromatic point sets in the plane by restricted orientation convex hulls. *Journal of Global Optimization*, 85(4):1003–1036, 2023. doi:10.1007/s10898-022-01238-9.

3 Edoardo Amaldi and Viggo Kann. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoretical Computer Science*, 147(1&2):181–210, 1995. doi:10.1016/0304-3975(94)00254-G.

4 Esther M. Arkin, Delia Garijo, Alberto Márquez, Joseph S. B. Mitchell, and Carlos Seara. Separability of point sets by k-level linear classification trees. *International Journal of Computational Geometry & Applications*, 22(2):143–166, 2012. doi:10.1142/S0218195912500021.

5 Esther M. Arkin, Ferran Hurtado, Joseph S. B. Mitchell, Carlos Seara, and Steven Skiena. Some lower bounds on geometric separability problems. *International Journal of Computational Geometry & Applications*, 16(1):1–26, 2006. doi:10.1142/S0218195906001902.

6 Bogdan Armaselu and Ovidiu Daescu. Dynamic minimum bichromatic separating circle. *Theoretical Computer Science*, 774:133–142, 2019. doi:10.1016/j.tcs.2016.11.036.

7 Boris Aronov, Delia Garijo, Yurai Núñez Rodríguez, David Rappaport, Carlos Seara, and Jorge Urrutia. Minimizing the error of linear separators on linearly inseparable data. *Discrete Applied Mathematics*, 160(10-11):1441–1452, 2012. doi:10.1016/j.dam.2012.03.009.

8 Timothy M. Chan. Low-dimensional linear programming with violations. *SIAM Journal on Computing*, 34(4):879–893, 2005. doi:10.1137/S0097539703439404.

9 Timothy M. Chan. Optimal partition trees. *Discrete & Computational Geometry*, 47(4):661–690, 2012. doi:10.1007/s00454-012-9410-z.

10 Bernard Chazelle and Emo Welzl. Quasi-optimal range searching in space of finite vc-dimension. *Discret. Comput. Geom.*, 4:467–489, 1989. doi:10.1007/BF02187743.

11 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition.* Springer, 2008.

12 Hazel Everett, Jean-Marc Robert, and Marc van Kreveld. An optimal algorithm for the ($\leq k$)-levels, with applications to separation and transversal problems. In *Proceedings of the ninth annual symposium on Computational geometry*, pages 38–46, 1993. doi:10.1145/160985.160994.

13 Erwin Glazenburg, Frank Staals, and Marc van Kreveld. Robust classification of dynamic bichromatic point sets in r2, 2024. arXiv:2406.19161, doi:10.48550/arXiv.2406.19161.

14 Erwin Glazenburg, Thijs van der Horst, Tom Peters, Bettina Speckmann, and Frank Staals. Robust bichromatic classification using two lines, 2024. arXiv:2401.02897, doi:10.48550/arXiv.2401.02897.

15 Sariel Har-Peled and Vladlen Koltun. Separability with outliers. In *Proc. 16th International Symposium on Algorithms and Computation*, volume 3827 of *Lecture Notes in Computer Science*, pages 28–39. Springer, 2005. doi:10.1007/11602613_5.

16 Ferran Hurtado, Mercè Mora, Pedro A. Ramos, and Carlos Seara. Separability by two lines and by nearly straight polygonal chains. *Discrete Applied Mathematics*, 144(1-2):110–122, 2004. doi:10.1016/j.dam.2003.11.014.

17 Ferran Hurtado, Marc Noy, Pedro A. Ramos, and Carlos Seara. Separating objects in the plane by wedges and strips. *Discrete Applied Mathematics*, 109(1-2):109–138, 2001. doi:10.1016/S0166-218X(00)00230-4.

18 Ferran Hurtado, Carlos Seara, and Saurabh Sethia. Red-blue separability problems in 3D. *International Journal of Computational Geometry & Applications*, 15(2):167–192, 2005. doi:10.1142/S0218195905001646.

19 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, 1984. doi:10.1145/2422.322418.

20 D. Sculley and Gabriel M. Wachman. Relaxed online SVMs for spam filtering. In *Proc. 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 415–422. Association for Computing Machinery, 2007. doi:10.1145/1277741.1277813.

21 Carlos Seara. *On geometric separability.* PhD thesis, Univ. Politecnica de Catalunya, 2002.

22 Aihua Shen, Rencheng Tong, and Yaochen Deng. Application of classification models on credit card fraud detection. In *Proc. 2007 International conference on service systems and service management*, pages 1–4. IEEE, 2007.